Introduction to Logistic Regression
Gives introduction to a machine learning algorithm: Logistic Regression. First, we describe the theoretical background of regression analysis using simple linear regression and the Generalized Linear Model. Then, we describe the Logistic Regression algorithm itself, and its solution using gradient descent. Finally, we provide an intuitive demonstration of how it works in a classification application with figures (including the MATLAB code used to generate the figures), and links to learn about more real-world applications.

Introduction

This is an introductory module on using Logistic Regression to solve large-scale classification tasks. In the first section, we will digress into the statistical background behind the generalized linear modeling for regression analysis, and then proceed to describe logistic regression, which has become something of a workhorse in industry and academia. This module assumes basic exposure to vector/matrix notation, enough to understand
Equation:

$$M = \begin{bmatrix} 2 & 2 \\ 1 & 0 \end{bmatrix}, x = \begin{bmatrix} 3 \\ -1 \end{bmatrix}, x_1 = ?, M*x = ?$$

What is all this about?

Regression Analysis is in essence the minimization of a cost function J that models the squared difference between the exact values y of a dataset, and one's estimate h of that dataset . Often, it is referred to as fitting a curve (the estimate) to a set of points based on some quantified measure of how well the curve fits the data. Formally, the most general form of the equation to model this process is:
Equation:

$$\begin{aligned} &\underset{x}{\text{minimize}} J\left(\theta\right) \\ &\text{subject to} \qquad\quad h_\theta\left(x\right) \end{aligned}$$

This minimization function models all regression analysis, but for the sake of understanding, this general form is not the most useful. How exactly do we model the estimate? How exactly do we minimize? To answer these questions and to be more specific, we shall begin by considering the simplest regression form, linear regression.

Linear Regression

In linear regression, we model the cost function's equation as:
Equation:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^i\right) - y^i\right)^2$$

What does this mean? Essentially, $h_\theta(x)$ is a vector that models one's hypothesis, the initial guess, of every point of the dataset. y is the exact value of every point in the dataset. Taking the squared difference between these two at every point creates a new vector that quantifies the error between one's guess and the actual value. We then seek to minimize the average value of this vector, because if this is minimized, then we have gotten our estimate to be as close as possible to the actual value for as many points as possible, given our choice of hypothesis.

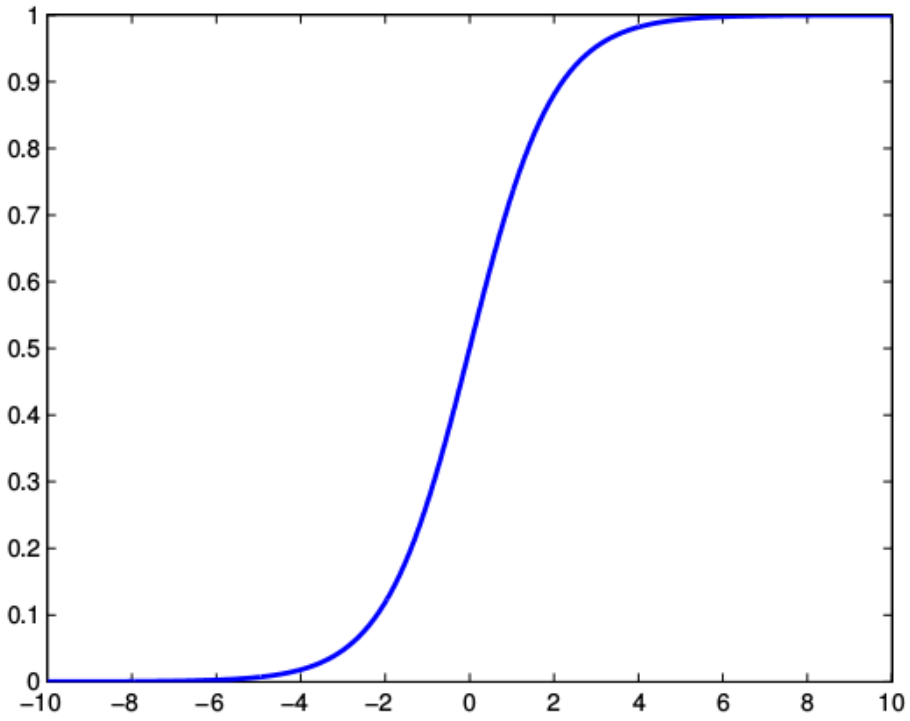
As the above module demonstrates, linear regression is simply about fitting to a line, whether that line is straight or contains an arbitrary number of polynomial features. But that hasn't quite gotten us to where we wanted to get, which is classification, so we may need more tools.

Generalized Linear Model

As was stated in the beginning, there are many ways to describe the cost function. In the above description, we used the simplest linear model that can describe the hypothesis, but there are a range of values that can go into the hypothesis, and they can be grouped into families of functions. We can construct a Generalized Linear Model to model these extensions systematically. We can describe the value of the estimate and the actual points by incorporating them inside of an exponential function. In our example, we shall use the sigmoid function, which is the following:
Equation:

$$g(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid Function



Sigmoid function for X in [-10,10]

Why might we want to do this? If we are doing classification between two different items, where one set of examples has the value 0 and one has the value 1, it doesn't make much sense to have values that are outside of this range. With linear regression, the predictions can take on any value. In order to model our hypothesis as being contained within the 0 to 1 range, we can use the sigmoid function. Using the Generalized Linear Model,

Equation:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta \left(x^i \right) - y^i \right)^2$$

Recall that in linear regression the hypothesis is

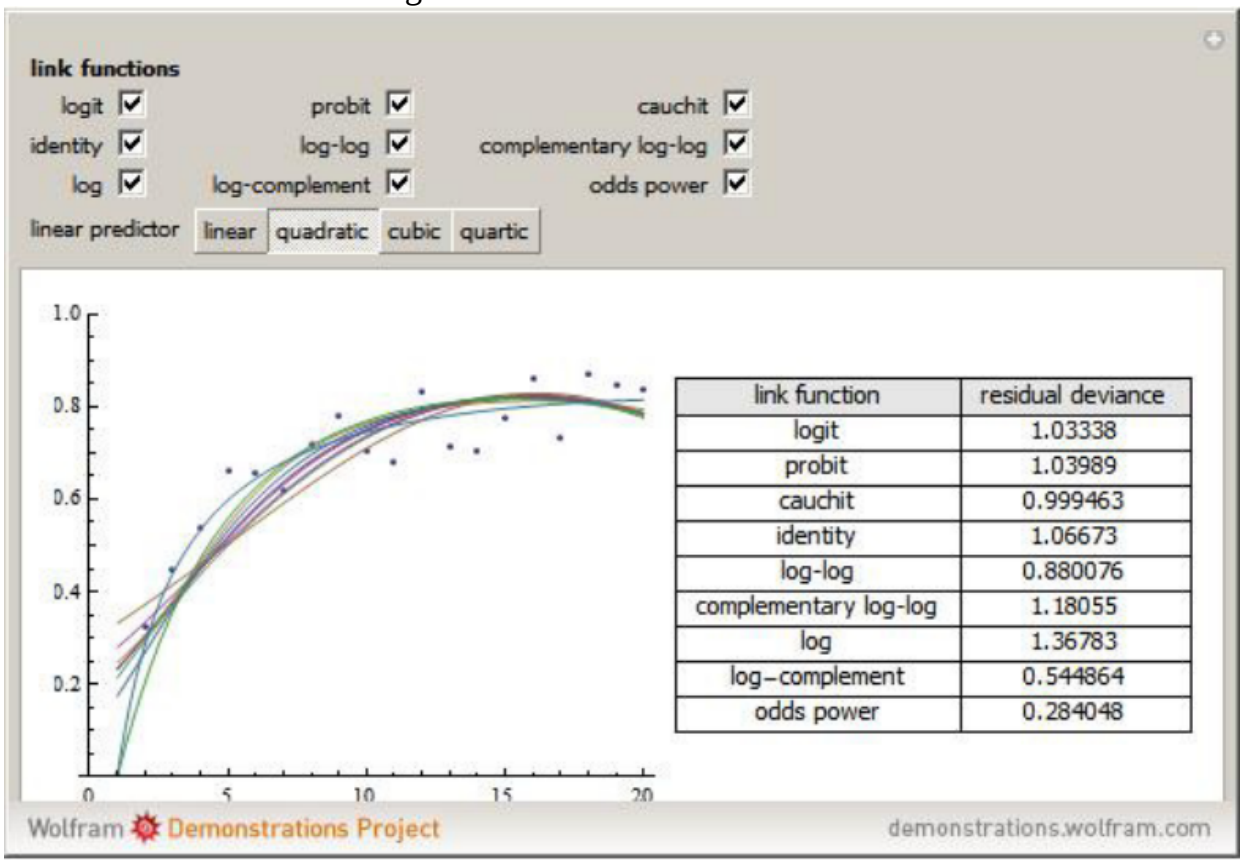
Equation:

$$h_\theta(x) = \theta^T x$$

In logistic regression, the hypothesis function is
Equation:

$$h_\theta(x) = g\left(\theta^T x\right) = \frac{1}{1 + e^{-\theta^T x}}$$

Wolfram Demonstrations Logit Function



http://demonstrations.wolfram.com/ComparingBinomialGeneralizedLinearModels/

Probabilistic Interpretation

What we are essentially doing with taking least-squares regression is fitting our data, but we can go about classifying by describing the probability boundary that one of our points is above and below a line, and finding the maximum likelihood estimate of a given theta.

If we define the Probabilities of being defined as class 1 or 0 as

Equation:

$$\begin{aligned} P(y=1|\theta) &= h_\theta\left(x\right) \\ P(y=0|\theta) &= 1-h_\theta\left(x\right) \end{aligned}$$

Then it becomes clear that the likelihoods are described as the following:
Equation:

$$L\left(\theta\right) = \prod_{i=1}^{m}\left(h_\theta\left(x^i\right)\right)^{y^i}\left(1-h_\theta\left(x^i\right)\right)^{1-y^i}$$

From statistics, it is well-known that taking the log of a maximum likelihood estimate will still achieve the same maximum, and calculating the log-likelihood is significantly easier from a computational standpoint. For a proof of this, see here .

We therefore take the log of the above cost function as a log-likelihood and obtain:
Equation:

$$l\left(\theta\right) = \sum_{i=1}^{m}\left(y^i log\left(h_\theta\left(x^i\right)\right) + \left(1-y^i\right) log\left(1-h_\theta\left(x^i\right)\right)\right)$$

Minimizing the Cost Function

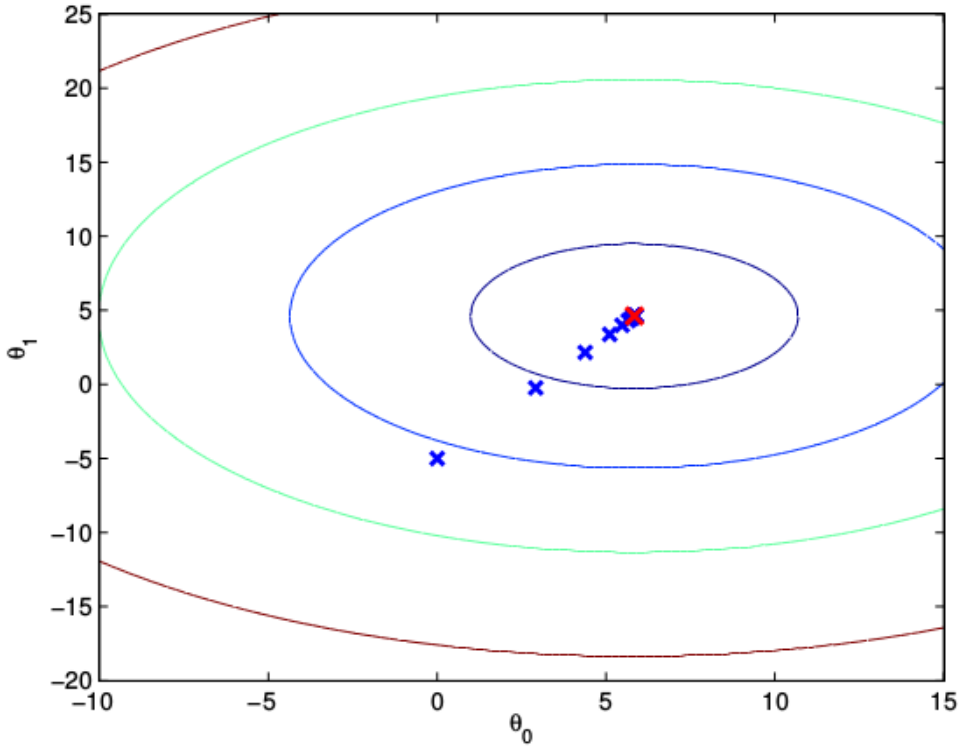
Now that we understand how we would classify these datasets exactly, how do we minimize the cost function? One simple way involves the application of Gradient Descent.

Gradient Descent is a method of approximating a cost function that gets you to converge to a final correct value. This is described for our cost function above as
Equation:

$$\theta_j = \theta_j - \alpha\frac{\partial}{\partial\theta_j}J\left(\theta\right)$$

That is, for every example, we update the theta value by subtracting (subject to some learning rate parameter α) the partial derivative of the cost function in terms of that example, and repeat until convergence.

If we plot the output of a gradient descent function, it will start at a random point on the contour plot, and then after every iteration, it will move closer to the optimal value.
Gradient Descent



Gradient Descent plot showing the trend towards the optimum value

Applying Logistic Regression

In this section, we apply logistic regression described in earlier section to a simulated data set and study how well it performs.

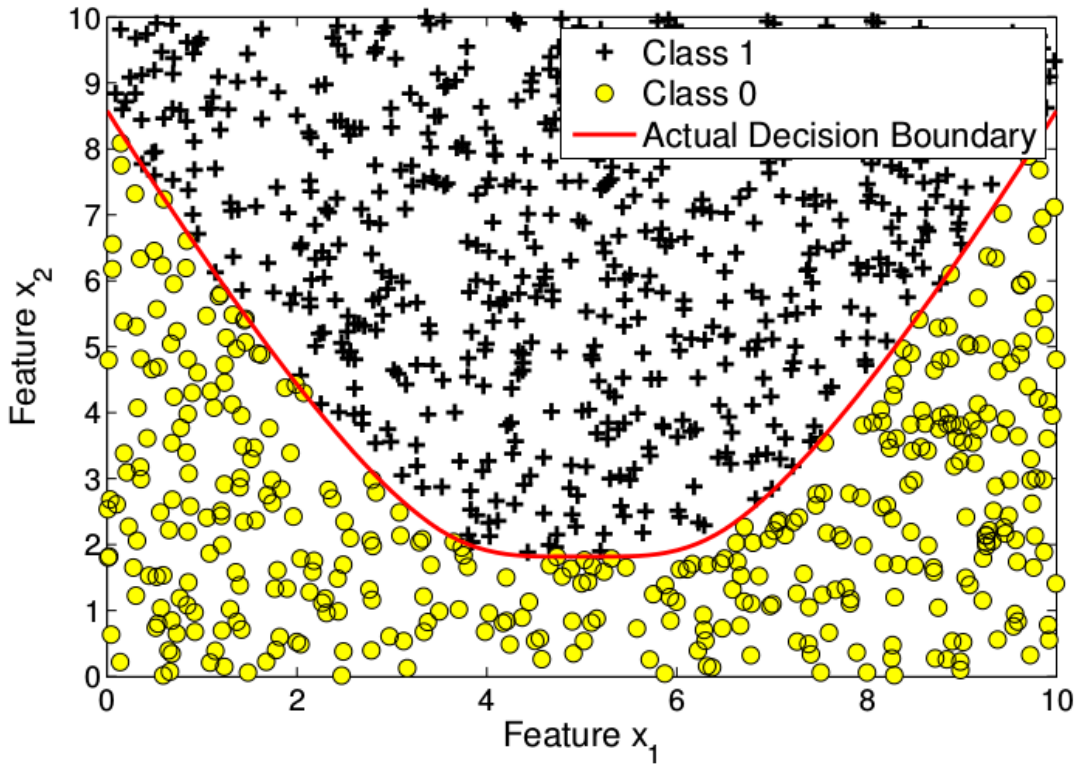
Data Generation

We simulated the case where each training example is described by two features x_1 and x_2. The two features x_1 and x_2 are generated uniformly in the range $[0, 10]$. The size of the training data set is 1000. The training examples were split into 2 classes, class 0 or class 1 based on the polynomial:
Equation:

$$(x_1 - 5)^4 - x_2^3 + 6 = 0$$

All the training examples that are above the polynomial eq. [link] belong to class 1 and the training examples below the polynomial curve belong to class 0. Notice that the true boundary that separates the two classes is a 4^{th} order polynomial.

The figure below plots the training data and the actual decision boundary.



Scatter Plot of Training Data

We split the training data set into 3 separate sets, training set with 600 examples, cross-validation set with 200 examples and test data set with 200 examples.

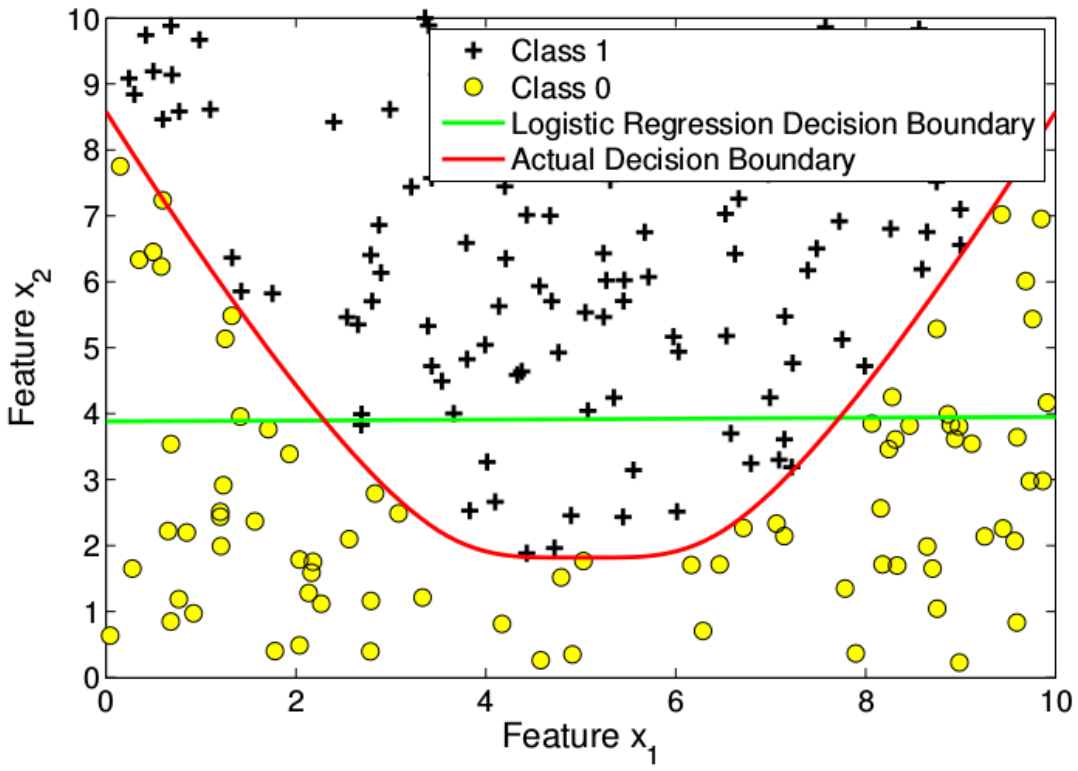
Classification

This training data was then fed into a logistic regression classifier to study the performance of the classifier. It is important to note that the objective of logistic regression classifier is maximizing the accuracy of labeling the data into two classes. Unlike linear regression, the decision boundary of the logistic regression classifier does not try to match the underlying true boundary which divides the data into two classes.

In addition to the two features that identify a training example, polynomial features up to a desired degree are generated. We start off the optimization with an initial parameter value of all 0. The optimization of the cost function is done using the Matlab's built-in *fminunc* function. A function *costFunctionReg.m* that outputs the regularized cost and regularized gradient, with the training data, parameter values and the regularization parameter as inputs, is given as input along with initial parameter values to this *fminunc* function.

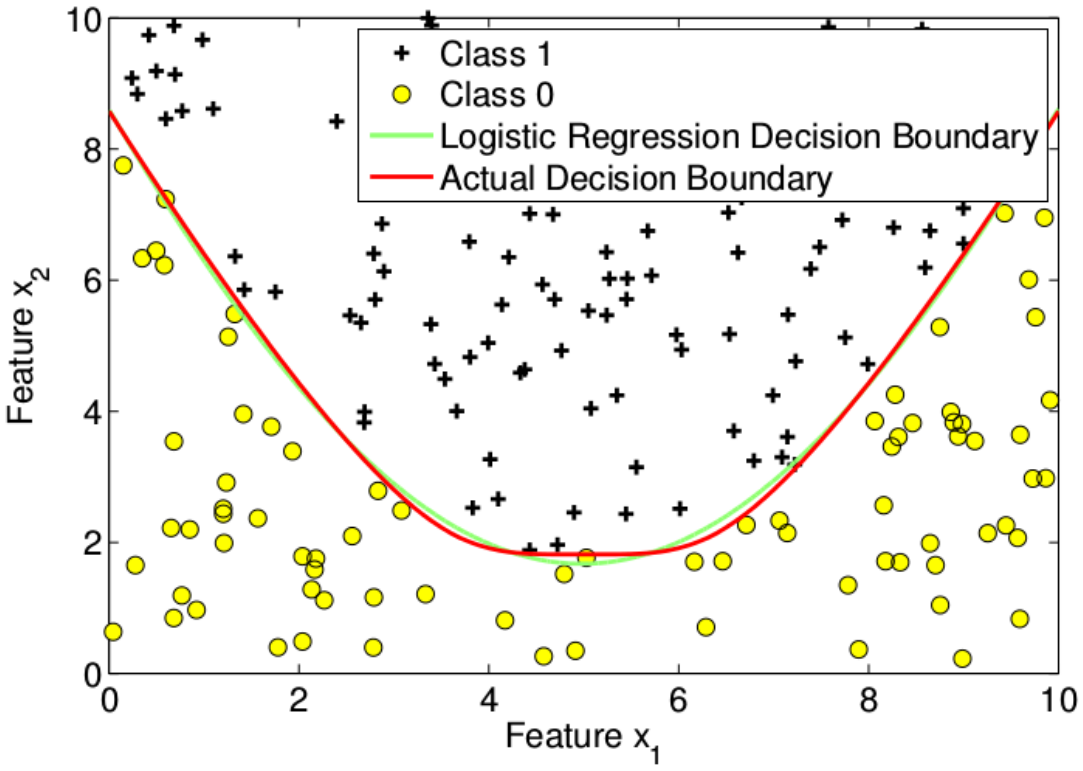
Now we vary the maximum degree of the polynomial features to study the decision boundary of the classifier. Its important to note that the values of the parameters are obtained from the training data set, for a given value of maximum degree. The optimal values of maximum degree are determined by the performance on the cross-validation set. Finally, the decision boundary obtained by solving for the parameters with optimal values of maximum degree is used to evaluate the performance on a test data set, in order to see how well the classifier generalizes.

The decision boundary for a degree 1 polynomial is shown in [link] below. The accuracy on the cross-validation set was 84.50.



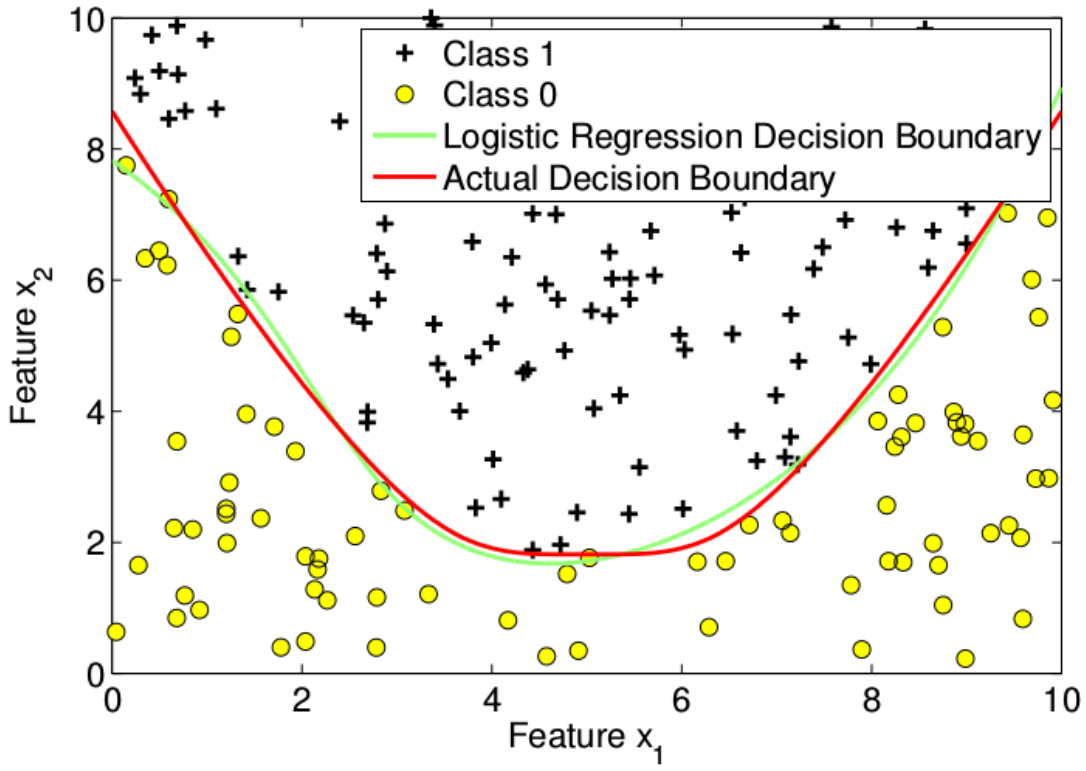
Logistic Regression with 1st degree features

From [link] it is clear that 1st degree features are not sufficient to capture both classes. So the maximum degree was increased to 2. [link] plots the decision boundary for degree 2. The accuracy of the classifier on cross-validation set in this case was 98.50.



Logistic Regression with 2nd degree features

We now try the maximum degree of 3. [link] plots the decision boundary with maximum degree 3. The accuracy on cross-validation set is 98.00.



Logistic Regression with 3rd degree features

Due to its lower accuracy, the logistic regression classifier with maximum degree 2 is chosen from amongst the 3 classifiers. This classifier was then evaluated on test data set to study how well it generalizes. The accuracy of this classifier on test set was 98.00. Hence this logistic regression classifier generalizes very well.

To see the MATLAB code that generated these plots, download the following .zip file: MATLAB files for simulated data. !

Conclusion

As was stated, this collection is intended to be an introduction to regression analysis, but is sufficient in order to understand the application of logistic regression to an application. There are plenty of resources to learn more about more nuanced views of the key components of the theory, and more resources to see logistic regression in action!

For an application of Logistic Regression to a synthetic dataset and to a real-world problem in statistical physics, see Optimizing Logistic Regression for Particle Physics !

Optimizing Logistic Regression for a Particle Physics Application
This is an application module describing using Logistic Regression to solve a detection problem in Particle Physics: Detecting the top quark.

Motivation

High-energy particle physics experiments today usually involves colliding beams of particles accelerated to tremendous energies and then studying the “shrapnel" that is created. The goal of such experiments often involves the discovery of new particles that are created when two common particles from the beams collide. Unfortunately, these new particles are so short-lived that they cannot be observed directly. Instead, the existence of such particles is inferred from patterns in the “shrapnel" that is formed when they decay into other particles. It is the properties of these secondary particles that are measured in detectors such as those at the Large Hadron Collider (LHC).

To discover a new particle, physicists use computer programs that have been programmed with theoretical models to simulate large numbers of random particle collisions. The models are tuned so as to generate two sets of collision “events": one set is compatible with the particle existing in nature and the other is compatible with the null hypothesis. The characteristics of the sets of simulated events are then compared with real events from a live experimental detector.

Because of the complicated nature of the events and the large amount of data involved, machine learning has been investigated to help with the classification problem of determining what kind of particles are initially produced for each event. If we can train classifiers on simulated data to be effective at distinguishing events that are associated with interesting particles from uninteresting background events, we will likely be able to use such classifiers to confirm or deny the discovery of these particles in real data. The existence of more effective classifiers decrease the amount of real data that must be collected to obtain a statistically significant result. Given the large expense of operating particle colliders and the power of modern computers, acquiring more effective classifiers is an important problem for particle physicists.

Project Goal

The project goal is to construct a classifier that is efficient at verifying the existence of a top quark in a set of simulated collision events. Top quarks are good particles for investigating classifiers because they are very rare and very hard to

detect but have already been proven to exist. Efficiency is defined by the beam luminosity needed to detect the existence of top quarks with 5σ of statistical significance, the “industry standard" for particle physics. Beam luminosity is directly proportional to the number of collisions, and thus the operating time of the experiment, and thus the cost, so lower required luminosity is better.

Top quarks can be created by a number of different pathways, and each pathway creates a different pattern of decay products measured by the detectors, real or simulated. This project focuses on only one particular pathway which results in the creation of a top quark and its anti-particle in what are known as “t-tbar" events.

Events which contain the desired t-tbar events are inevitably accompanied by a much larger number of undesired background events. The vast majority of these events initially involve the creation of much lighter quarks – these are called “QCD" events, named after the theory that describes the behavior of these particles, quantum chromodynamics. QCD events are several million times more common than t-tbar events at the energies presently used at the LHC. A minority of background events involve the creation of W bosons. Although these W events are rarer, occurring at the rate of “only" a few hundred per t-tbar event, their detectable features are very similar to those of t-tbar events and are thus they are harder to distinguish from the desired t-tbar signal.

Prior Work - PHYS 491 Summer 2011

The author initially got started investigating this classification problem as part of an independent study course taken with Dr. Paul Padley. Dr. Padley works at Bonner Lab at Rice University and is manager of the Endcap Muon Subdetector, a large component of the Compact Muon Solenoid (CMS) experiment at the LHC. The course involved learning background about particle physics necessary to be able to understand the classification problem.

The strategy over the summer was to figure out how to use a popular event generating program called Pythia in conjunction with a popular machine learning toolkit called WEKA. The author wrote a small program in C to interface with Pythia to generate a large number of t-tbar, W, and QCD events. The C program ran the events through a simple filter, a “trigger" in physics parlance, to quickly eliminate a large number of events that had a relatively low probability of being t-tbar events. (The trigger filters out 95% of t-tbar events, but 99.9% of the QCD

background events.) Sufficient data was collected to form a training data set and a test data set; each set had 10,000 of each type of event.

Features

When Pythia generates an event, it makes available a wide array of information useful for generating features. The chosen features for use in this project included how many of each type of lepton (electrons, muons, and tau particles) were created in each event, how many “jets" corresponding to quarks were generated, and the minimum angle between any pair of quark jets. In addition, missing transverse momentum, indicative of an invisible neutrino, was measured. Finally, the total transverse energy (energy perpendicular to the beam axis) of all of the quark jets was measured. A large transverse energy is strongly associated with “head-on" collisions capable of releasing enough energy to make top quarks. “It turns out" that transverse energy is the most important single feature for identifying top quarks.

Classifiers

Because of the author's lack of experience with machine learning, a script was developed that ran each of the classifiers in WEKA against the training data and extracted the relevant statistics. Each of the classifiers was run with default parameters. For each classifier, the number of t-tbar events that were correctly identified (true positives) was determined, as well as the number of QCD and W events that were misclassified as t-tbar events (false positives.) From the ratio of these numbers and the cross-sections of the relevant pathways, it was possible to determine the total beam luminosity needed to confirm the existence of top quarks in a set of events using each classifier.

Most of the wide variety of classifiers performed within an order of magnitude of each other, except for one called “Hyperpipes" which performed two orders of magnitude better than any of the others. This struck Dr. Padley as very strange as he had never heard of the Hyperpipes algorithm before.

Current Work - ELEC 631 Fall 2011

One interesting aspect of the behavior of the Hyperpipes algorithm on the dataset was the very low true positive rate: less than 10% of the t-tbar events were correctly classified as such. However, the false positive rate was much, much lower – this is what accounted for its good performance. At this point the author became suspicious that the default parameters for the WEKA machine learning algorithms were poorly suited to this application and that some of the other algorithms could also demonstrate greatly improved performance with a little tuning. Since logistic regression is relatively simple, and since the author now understands a little about how the algorithm works from the on-line Stanford course, the decision was made to try and tune it to this application.

Cost Matrix

The first stage of tuning involved adjusting the cost matrix. This followed from the observation that when training a classifier, we really want to penalize false positives much more harshly than false negatives. Sacrificing up to 50% of our true positives is acceptable if it leads to a very large decrease in background false positive events and thus an improved signal-to-noise ratio. Keeping the true positive rate relatively high is still important as t-tbar events are rare enough that keeping enough for statistical significance is still a problem, as we shall see a bit later.

A variety of cost matrices were created and the logistic regression classifier in WEKA was trained on a training data set with each one of them. The resulting classification models were then tested on a cross-validation data set. Both the training data set and the cross-validation set had 10,000 of each type of event as before. The results are summarized in Table TODO. Note that as expected, increasing the cost of false positives relative to other kinds of errors greatly improved the signal to noise ratio with an acceptable decrease in the true positive rate. Note that the cost matrices are properly scaled by WEKA: only the relative costs of the different types of errors represented by the cost matrix matter. Interestingly, eliminating the costs associated with misclassifying W background events as QCD background events and vice-versa seems to have no effect on the signal-to-noise ratio.

Choosing a cost matrix which penalizes each type of false positive 300 times more than other types of errors seemed to give good results without leading to diminishing returns in the form of decreased true positives. Penalizing false

positives caused by QCD events more than W events didn't improve the results on the large test set.

Ratio of Signal to Background in Training Set

Dr. Devika Subramanian of the Rice Computer Science department suggested that classifiers need to be trained on data that has each type of event to be classified in a ratio that is roughly the same as that in the real test data. This suggestion indicated that the previous strategy of training classifiers with an equal ratio of each type of event was unsound. Unfortunately, generating training and test sets with the correct ratio of event types is quite difficult because of the huge backgrounds that must be generated, in this case several million QCD events for every t-tbar event. Generating enough background for 100 t-tbar events would take a modern cpu core over 1000 days to compute enough background.

The solution was to use the DaVinci cluster at Rice to generate the backgrounds by running up to 200 Pythia simulation runs at once. This process generated enough background for 12 t-tbar events and this served as our test set. Unfortunately signal-to-noise ratio in the test set was so low that the logistic regression algorithm in WEKA could not be trained against it – the resulting model failed to detect any t-tbar events at all. Also, training logistic regression on the full test set strained the memory capabilities of the Java virtual machine it was run on, leading to frequent crashes.

To get around this, the author decided to compromise and experiment with training the logistic regression classifier on much smaller training sets with differing ratios of t-tbar events to background events. The performance of the resulting models were then tested on the cross-validation set containing 10,000 of each type of event. The model generated from a training set with a ratio of tt-bar to background events of about 50 seemed to perform the best. (See Table TODO).

Results

As a result of performing the above optimizations, the efficiency of the logistic regression model at analyzing the very large test set improved by over a factor of 30 while only halving the true positive rate, as shown in Table TODO. It is important to note that the false positive ratio is still much too high for top quarks to be discoverable with a data set of this size. Since we are dealing with counting

statistics of independent events, the uncertainty in the background count is approximately the square root of the background count, corresponding to a standard deviation of about 13. Since our signal is only 6 t-tbar events, this means we have a signal significance of about 0.4σ. In order to get a statistically significant result, we would need to collect around 150 times as much raw data.

Conclusion

We have demonstrated that we can optimize our use of linear regression to exploit characteristics of a particular particle physics data set. Existing tools like WEKA make using machine learning for this task relatively straightforward, with no need to reinvent the wheel.

It should be noted that this project has neglected the most difficult and computationally intensive part of identifying new physics with particle detectors: modeling the performance of the particle detectors themselves. Modern particle detectors are incredibly complicated pieces of machinery and modeling their capabilities (which change often as components are upgraded) requires a measurable fraction of the planet's computing resources. (ref Grid Computing)

Future Work

Dr. Subramanian also suggested that classifier performance could be improved by combining several integer features, namely how many of each type of lepton were found in each event, into one category feature, namely which lepton type was found. This makes sense because the high-level trigger eliminates all events that do not have exactly one lepton. A simple script should be able to transform all of the existing data to make this possible.

References

WEKA Pythia Particle physics book

$http://www.readwriteweb.com/archives/cern_officially_unveils_its_gr.php$

Acknowledgments

The author would like to thank Dr. Paul Padley and Dr. Devika Subramanian for providing advice and training for this project, as well as Dr. Andrew Ng for his excellent and fun on-line machine learning class.

Directions for Using Code

Install Pythia 8 and WEKA on your UNIX machine. The included scripts and Makefile assume that the WEKA classes are in /usr/share/java/weka.jar and that the directory containing the code and data files is located in the pythia directory. See the included README file for more details.